CSC 344 Fourth Racket Programming Assignment Solution

Learning Abstract: This programming assignment features some recursive functions that make use of some basic list processing functions like append, car, cdr, and cons. The latter tasks also make use of some higher level functions like map, filter, and foldr.

First Task: Generate Uniform List

> Code

```
(require 2htdp/image)
(define (generate-uniform-list number element)
  (cond ((= number 0)
         '())
        ((> number 0)
     (append(list element)(generate-uniform-list (- number 1) element))
    )
  )
```

> Demo

```
> (generate-uniform-list 5 'kitty)
'(kitty kitty kitty kitty kitty)
> (generate-uniform-list 10 2)
'(2 2 2 2 2 2 2 2 2 2)
> (generate-uniform-list 0 'whatever)
'()
> (generate-uniform-list 2 '(racket prolog haskell rust))
'((racket prolog haskell rust) (racket prolog haskell rust))
```

Second Task: Association List Generator

> Code

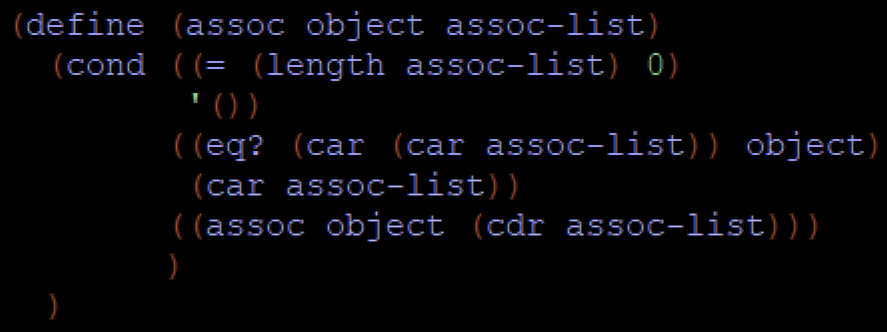
```
(define (a-list objects more-objects)
  (cond ((= (length objects) 0)
         '())
        ((> (length objects) 0)
         (cons (cons (car objects)(car more-objects))
                     (a-list (cdr objects)(cdr more-objects))
                     )
         )
        )
  )
```

> Demo

```
> (a-list '(one two three four five) '(un deux trois quatre cinq))
'((one . un) (two . deux) (three . trois) (four . quatre) (five . cinq))
> (a-list '() '())
'()
> (a-list '(this) '(that))
'((this . that))
> (a-list '(one two three) '((1)(2 2)(3 3 3)))
'((one 1) (two 2 2) (three 3 3 3))
```

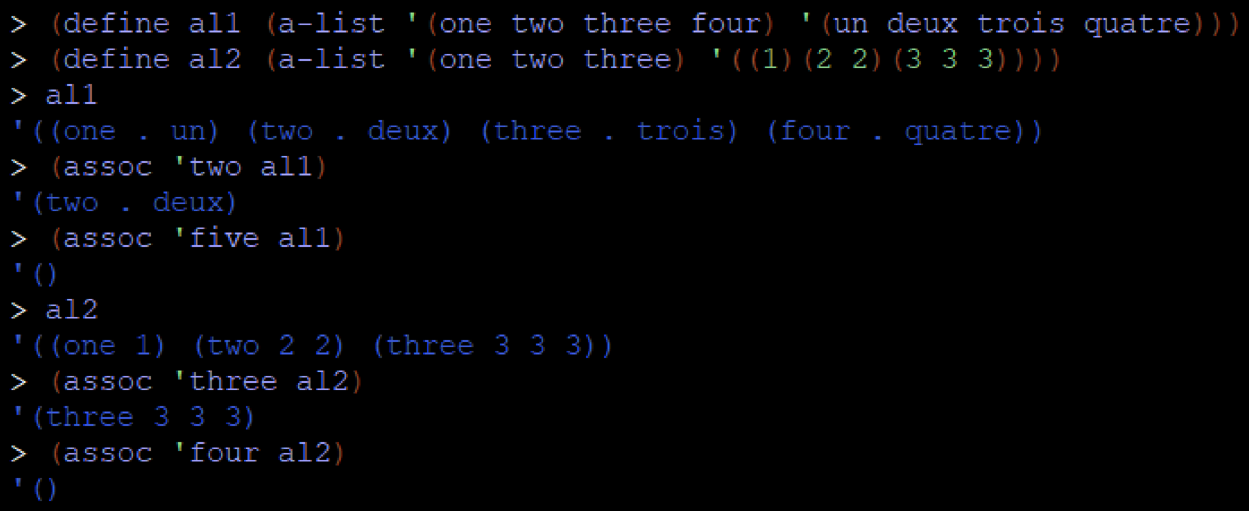
Third Task: Assoc

> Code



```
(define (assoc object assoc-list)
  (cond ((= (length assoc-list) 0)
          '())
        ((eq? (car (car assoc-list)) object)
         (car assoc-list))
        ((assoc object (cdr assoc-list)))
        )
  )
```

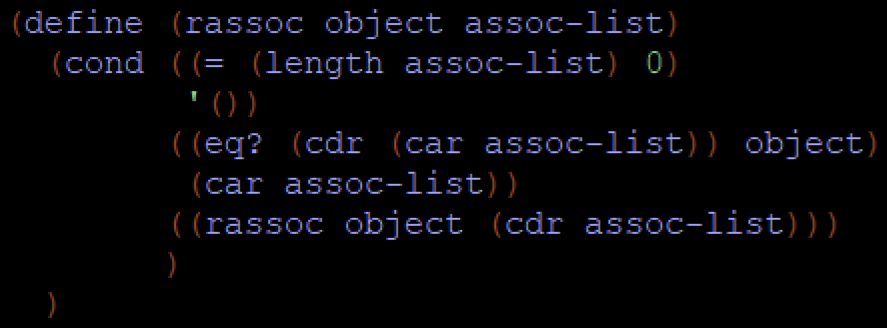
> Demo



```
> (define al1 (a-list '(one two three four) '(un deux trois quatre)))
> (define al2 (a-list '(one two three) '((1)(2 2)(3 3 3))))
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> (assoc 'two al1)
'(two . deux)
> (assoc 'five al1)
'()
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (assoc 'three al2)
'(three 3 3 3)
> (assoc 'four al2)
'()
```

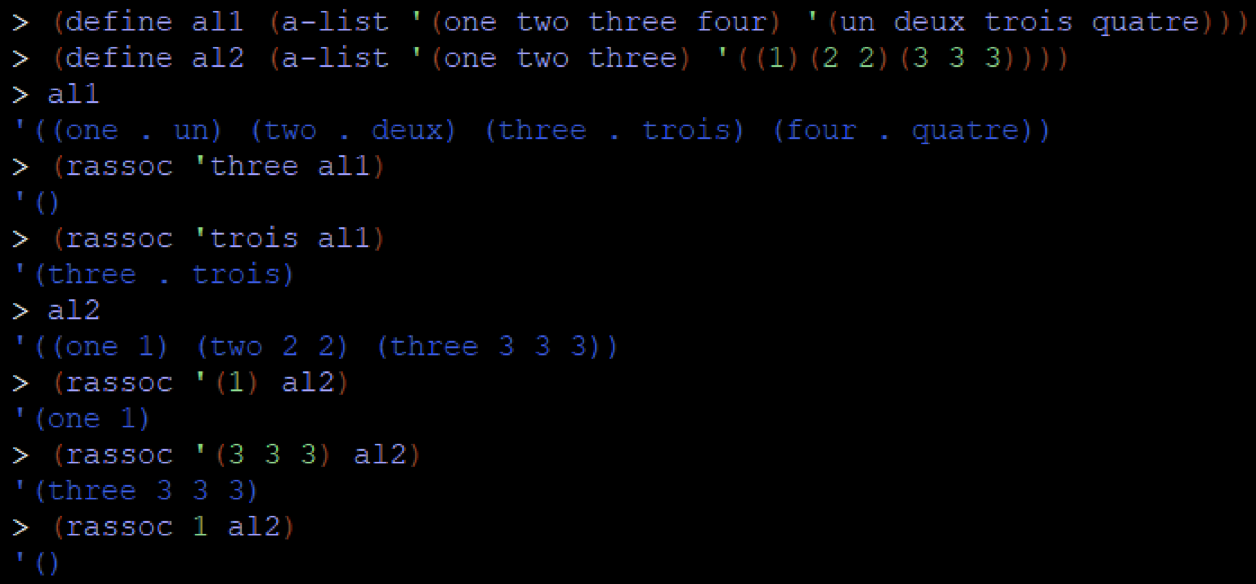
Fourth Task: Rassoc

> Code



```
(define (rassoc object assoc-list)
  (cond ((= (length assoc-list) 0)
          '())
         ((eq? (cdr (car assoc-list)) object)
          (car assoc-list))
         ((rassoc object (cdr assoc-list)))
         )
  )
```

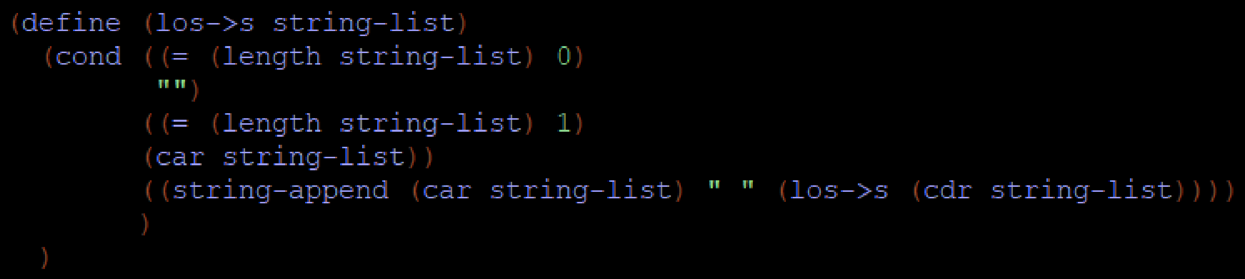
> Demo



```
> (define al1 (a-list '(one two three four) '(un deux trois quatre)))
> (define al2 (a-list '(one two three) '((1)(2 2)(3 3 3))))
> al1
'((one . un) (two . deux) (three . trois) (four . quatre))
> (rassoc 'three al1)
'()
> (rassoc 'trois al1)
'(three . trois)
> al2
'((one 1) (two 2 2) (three 3 3 3))
> (rassoc '(1) al2)
'(one 1)
> (rassoc '(3 3 3) al2)
'(three 3 3 3)
> (rassoc 1 al2)
'()
```

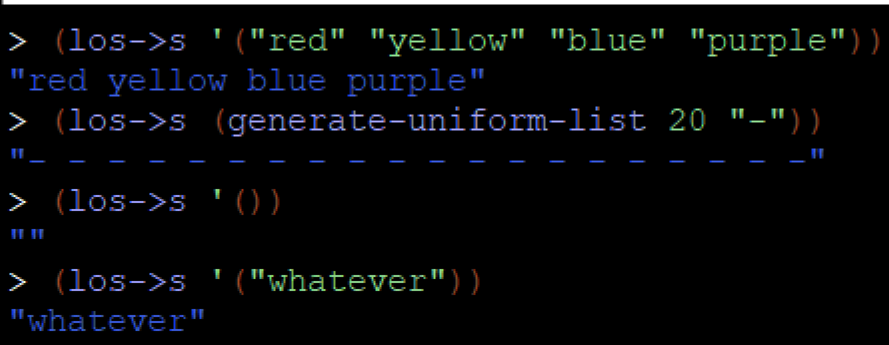
Fifth Task: Los->s

> Code



```
(define (los->s string-list)
  (cond ((= (length string-list) 0)
         "")
        ((= (length string-list) 1)
        (car string-list))
        ((string-append (car string-list) " " (los->s (cdr string-list))))
        )
  )
```

> Demo



```
> (los->s '("red" "yellow" "blue" "purple"))
"red yellow blue purple"
> (los->s (generate-uniform-list 20 "-"))
"- - - - - - - - - - - - - - - - - - - -"
> (los->s '())
""
> (los->s '("whatever"))
"whatever"
```

Sixth Task: Generate List

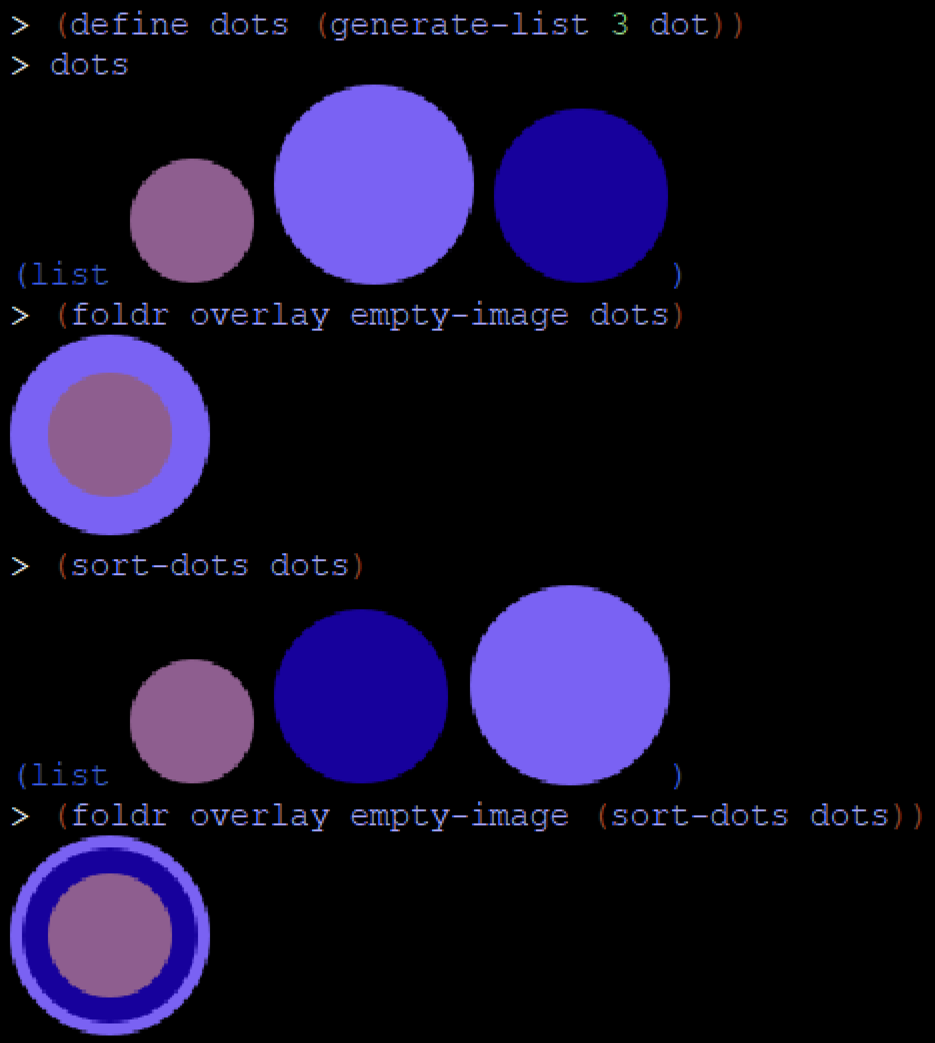
> Code

```
(define (big-dot)
  (circle (+ 10(random 81)) "solid" (random-color))
  )

(define (dot)
  (circle (+ 10(random 41)) "solid" (random color))
  )

(define (random-color)
  (color (rgb-value)(rgb-value)(rgb-value))
  )

(define (rgb-value)
  (random 256)
  )

(define (sort-dots loc)
  (sort loc #:key image-width <)
  )

(define (generate-list int function)
  (cond ((= int 0)
         '())
        ((> int 0)
         (cons (function)(generate-list(- int 1) function)))
        )
  )
```

> Demo 1

```
> (generate-list 10 roll-die)
'(6 2 1 1 3 4 3 5 3 2)
> (generate-list 20 roll-die)
'(6 3 5 6 2 6 2 5 4 1 4 1 2 5 2 6 1 4 3 1)
> (generate-list 12 (lambda()(list-ref '(red yellow blue)(random 3))))
'(blue yellow blue blue yellow yellow yellow red blue blue red blue)
```

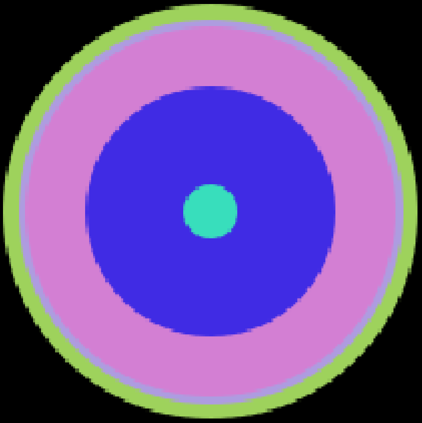
> Demo 2



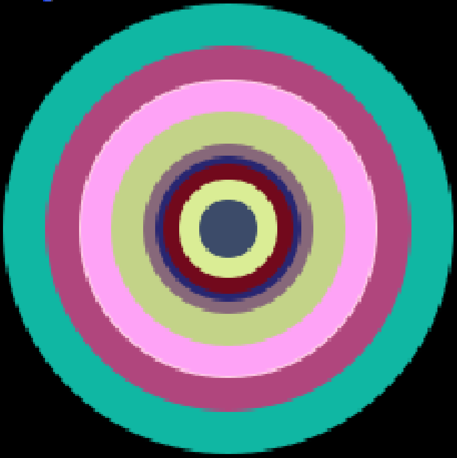
```
> (define dots (generate-list 3 dot))
> dots
(list                                      )
> (foldr overlay empty-image dots)

> (sort-dots dots)
(list                                      )
> (foldr overlay empty-image (sort-dots dots))
```

> Demo 3

```
> (define a (generate-list 5 big-dot))
> (foldr overlay empty-image (sort-dots a))
```



```
> (define b (generate-list 10 big-dot))
> (define a (generate-list 5 big-dot))
> (foldr overlay empty-image (sort-dots b))
#<procedure:>>
```



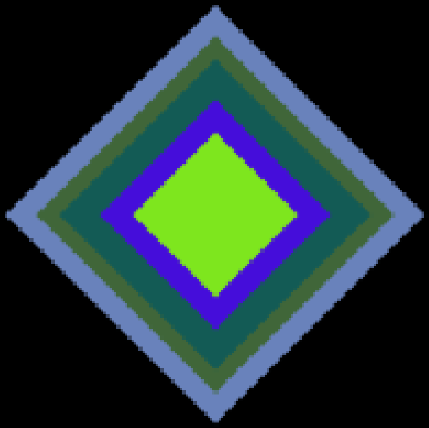
Seventh Task: The Diamond

> Code

```
(define (create-diamond)
  (rotate 45(square (+ 30(random 100)) "solid" (random-color)))
  )

(define (sort-diamonds loc)
  (sort loc #:key image-width <)
  )

(define (diamond amount)
  (foldr overlay empty-image (sort-diamonds (generate-list amount create-diamond)))
  )
```

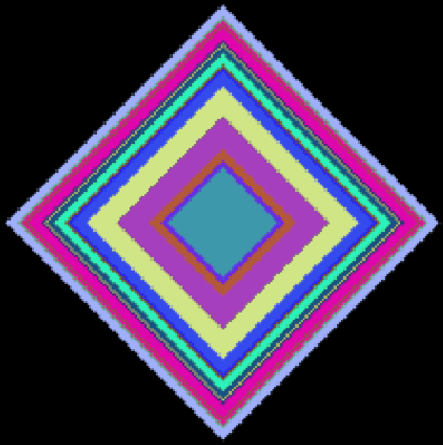
> Demo 1

```
> (diamond 5)
```



> Demo 2

```
> (diamond 20)
```



Eighth Task: Chromesthetic Renderings

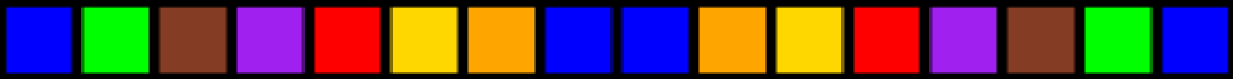
> Code

```
(define pitch-classes '(c d e f g a b))

(define color-names '(blue green brown purple red yellow orange))

(define (box color)
  (overlay (square 30 "solid" color)
           (square 35 "solid" "black"))
  )

(define boxes (list (box "blue")
                    (box "green")
                    (box "brown")
                    (box "purple")
                    (box "red")
                    (box "gold")
                    (box "orange"))
  )
```

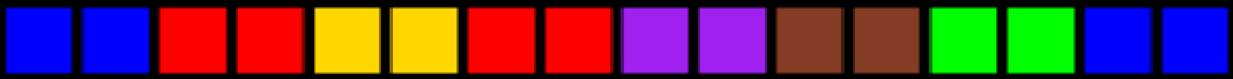
```
(define pc-a-list (a-list pitch-classes color-names))

(define cb-a-list (a-list color-names boxes))

(define (pc->color pc)
  (cdr (assoc pc pc-a-list))
  )

(define (color->box color)
  (cdr (assoc color cb-a-list))
  )

(define (play pitches)
  (cond ((empty? pitches)
         empty-image)
        (else
         (define pitch (car pitches))
         (define pitch-color (cdr (assoc pitch pc-a-list)))
         (define pitch-box (cdr (assoc pitch-color cb-a-list)))
         (beside pitch-box (play (cdr pitches)))
         )
        )
  )
```

> Demo

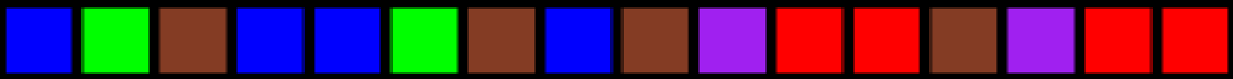
```
> (play '(c d e f g a b c c b a g f e d c))
```



```
> (play '(c c g g a a g g f f e e d d c c))
```



```
> (play '(c d e c c d e c e f g g e f g g))
```



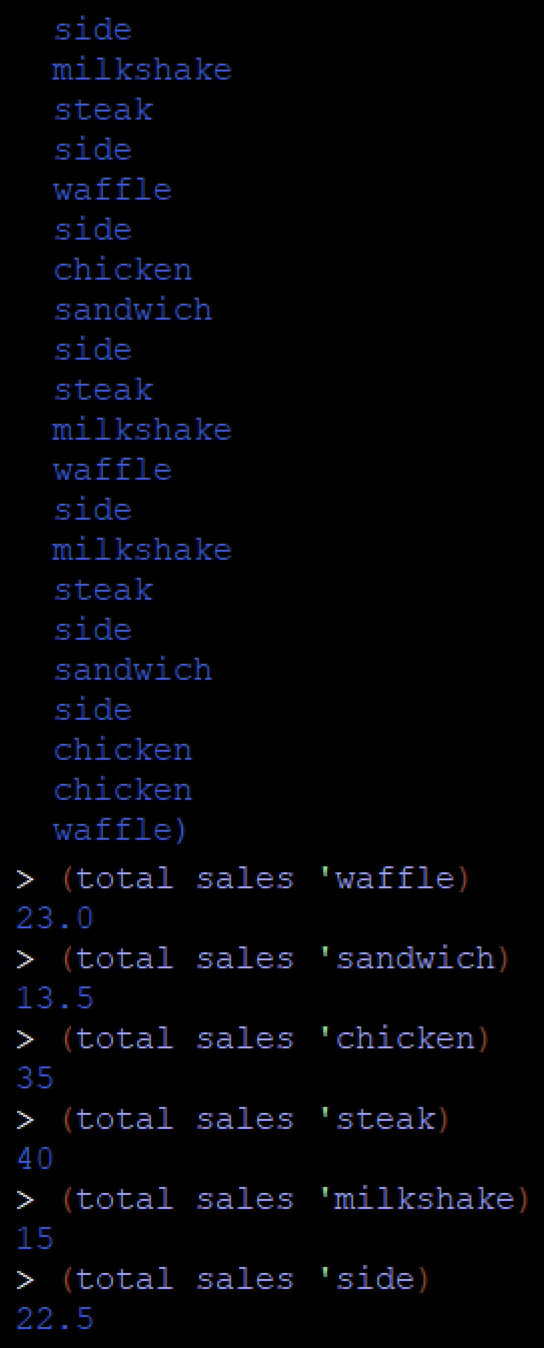
Ninth Task: Diner

> Code

```
(define menu '((waffle . 5.75)(sandwich . 4.5)(chicken . 7)(steak . 10)
               (milkshake . 3)(side . 2.5)))

(define sales '(chicken side steak milkshake sandwich side waffle milkshake chicken side
                milkshake steak side waffle side chicken sandwich side steak milkshake
                waffle side milkshake steak side sandwich side chicken chicken waffle))

(define (item-price order)
  (cdr (assoc order menu))
  )

(define (total orders item)
  (define prices (map item-price (filter (lambda (o)(eq? o item)) sales)))
  (foldr + 0 prices)
  )
```

> Demo

```
> menu
'((waffle . 5.75)
  (sandwich . 4.5)
  (chicken . 7)
  (steak . 10)
  (milkshake . 3)
  (side . 2.5))
> sales
'(chicken
  side
  steak
  milkshake
  sandwich
  side
  waffle
  milkshake
  chicken
```



```
  side
  milkshake
  steak
  side
  waffle
  side
  chicken
  sandwich
  side
  steak
  milkshake
  waffle
  side
  milkshake
  steak
  side
  sandwich
  side
  chicken
  chicken
  waffle)
> (total sales 'waffle)
23.0
> (total sales 'sandwich)
13.5
> (total sales 'chicken)
35
> (total sales 'steak)
40
> (total sales 'milkshake)
15
> (total sales 'side)
22.5
```

Tenth Task: Wild Card

> Specification

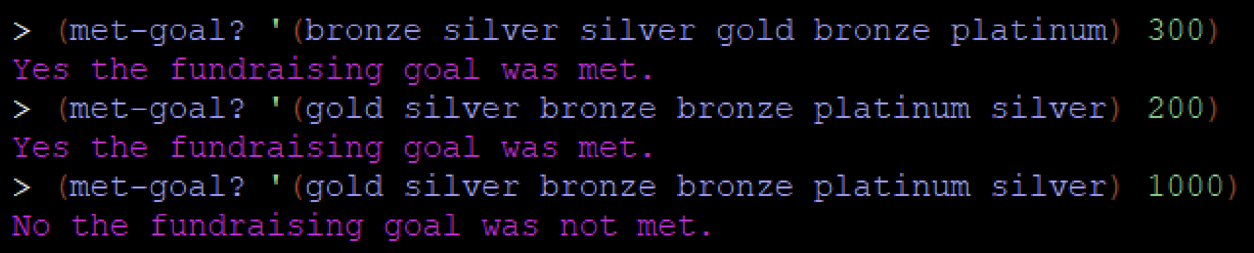
Many institutions have sponsorship programs in which sponsors can pay a certain amount which will then give them a ranking (In this case, Bronze - $50, Silver - $100, Gold - $150, and Platinum - $200). This ranking gives the sponsor certain rewards that get better as the rank gets higher. My goal was to write a function that would calculate whether or not any certain amount and combination of sponsors would meet a fundraising goal. My function includes:

1. The first parameter is a list of sponsors represented by their levels.
2. The second parameter is an integer that represents the fundraising goal.
3. The function outputs whether or not the goal was reached.

> Code

```scheme
(define sponsor-levels '((bronze . 50)(silver . 100)(gold . 150)(platinum . 200)))

(define (sponsor-price level)
  (cdr (assoc level sponsor-levels))
  )

(define (met-goal? sponsors goal)
  (define amount-raised (map sponsor-price sponsors))
  (define total(foldr + 0 amount-raised))
  (cond ((>= total goal)
         (display "Yes the fundraising goal was met."))
        (else
         (display "No the fundraising goal was not met."))
        )
  )
```

> Demo



```
> (met-goal? '(bronze silver silver gold bronze platinum) 300)
Yes the fundraising goal was met.
> (met-goal? '(gold silver bronze bronze platinum silver) 200)
Yes the fundraising goal was met.
> (met-goal? '(gold silver bronze bronze platinum silver) 1000)
No the fundraising goal was not met.
```